

# DISTRIBUTED OS- UNIT 3

## CLOCK SYNCHRONIZATION

**(Clock Synchronization to Threads)**

*Dr. K. Geetha, Associate Professor,  
Department of Computer Science,  
Periyar Arts College, Cuddalore*

# UNIT III CLOCK SYNCHRONIZATION

**Unit III** -Clock Synchronization – Event Ordering  
– Mutual Exclusion – Deadlock – Election  
Algorithms - Process Migration – Threads.

**Reference:** Distributed Operating Systems  
Concepts and Design- Pradeep K. Sinha, PHI,  
New Delhi, 2007.

# SYNCHRONIZATION

- Sharing of resources is Economical.
- Sharing may be cooperative or competitive  
examples of process cooperation
  - producer-consumer or
  - client-server relationship.

Sharing needs synchronization

# SYNCHRONIZATION

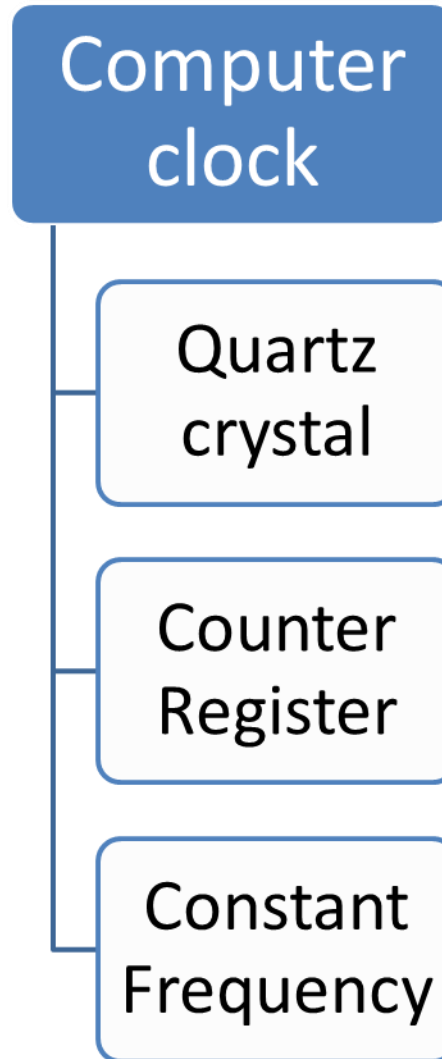
- Synchronization-related issues
  - Clock synchronization
  - Event ordering
  - Mutual exclusion
  - Deadlock
  - Election algorithms

# CLOCK SYNCHRONIZATION

- Timer mechanism (called a computer clock)
  - to keep track of current time
  - for various accounting purposes such as calculating the time spent by a process in CPU utilization, disk , and so on, so that the corresponding user can be charged properly.

# CLOCK SYNCHRONIZATION

## How Computer Clocks Are Implemented



# CLOCK SYNCHRONIZATION

- To make the computer clock function as an ordinary clock life, the following things are done:
  - 1 . The value in the constant register is chosen so that 60 clock ticks occur in a second.
  2. The computer clock is synchronized with real time (external clock).

For this, two more values are stored in the system  
-a fixed starting date and time and the  
number of ticks.

# DRIFTING OF CLOCKS

A clock always runs at a constant rate because its quartz crystal oscillates at a well-defined frequency

The difference accumulated over many oscillations leads to an observable difference in the times of the two clocks

clocks do not always maintain perfect time



# DRIFTING OF CLOCKS

suppose that when the real time is  $t$ , the time value of a clock

$p$  is  $C_p(t)$ .

If all clocks in the world were perfectly synchronized, we would have  $C_p(t) = t$  for all  $p$  and all  $t$ .

That is, if  $C$  denotes the time value of a clock, in the ideal case

$dc/dt$  should be 1.

$1-p \leq dc/dt \leq 1+p$

# DRIFTING OF CLOCKS

The nodes of a distributed system must periodically resynchronize

Their local clocks to maintain a global time base across the entire system

# DRIFTING OF CLOCKS

Distributed system requires the following types of clock synchronization

1. Synchronization of the computer clocks with real-time (or external) clocks
2. Mutual (or internal) synchronization of the clocks of different nodes of the system:

# CLOCK SYNCHRONIZATION ISSUES

No two clocks can be perfectly synchronized.

The difference in time values of two clocks is called clock skew.

clock skew of any two clocks is set less than  $\delta$ .

Clock synchronization requires each node to read the other nodes' clock values.

Errors occur mainly because of unpredictable communication delays

# CLOCK SYNCHRONIZATION ISSUES

Important issue

Time running backward

clock should be slowed down instead of  
adjustment

# CLOCK SYNCHRONIZATION ALGORITHMS

Centralized and  
Distributed algorithms

Centralized Algorithms:

one node has a real-time receiver (time server node)

clock time of this node is regarded as correct and used as the reference time.

clocks of all other nodes synchronized with the clock time of the time server node.

# CLOCK SYNCHRONIZATION ALGORITHMS

Depending on the role of the time server node, centralized clock synchronization algorithms are again of two types

Passive time server and

Active time server

# CENTRALIZED ALGORITHMS

Passive Time Server:

The time server receives the message,

It quickly responds with a message ("time = T")

- when the client node sends the "time=?" message, its clock time is  $T_0$
- when it receives the "time = T" message, its clock time is  $T_1$ .



# CENTRALIZED ALGORITHMS

$T_0$  and  $T_1$  are measured using the same clock,

The best estimate of the time required for the propagation of the message

"time =  $T$ " from the time server node to the client's node is  $(T_1 - T_0)/2$ .

The reply is received at the client's node, its clock is readjusted to  $T + (T_1 - T_0)/2$ .

# CENTRALIZED ALGORITHMS

## Active Time Server Centralized Algorithm

In the passive time server approach, the time server only responds to requests for time from other nodes.

In the active time server approach, the time server periodically broadcasts its clock time ("time = T").

# CENTRALIZED ALGORITHMS

Each node has prior knowledge of the approximate time ( $T_0$ ) required for the propagation of the message "time =  $T$ " from the time server node to its own node.

Drawback of this method

- ❖ Not fault tolerant.
- ❖ Chance for a node readjusted to an incorrect value.
- ❖ Requires broadcast facility

# CENTRALIZED ALGORITHMS

Active time server algorithm is the Berkeley algorithm.

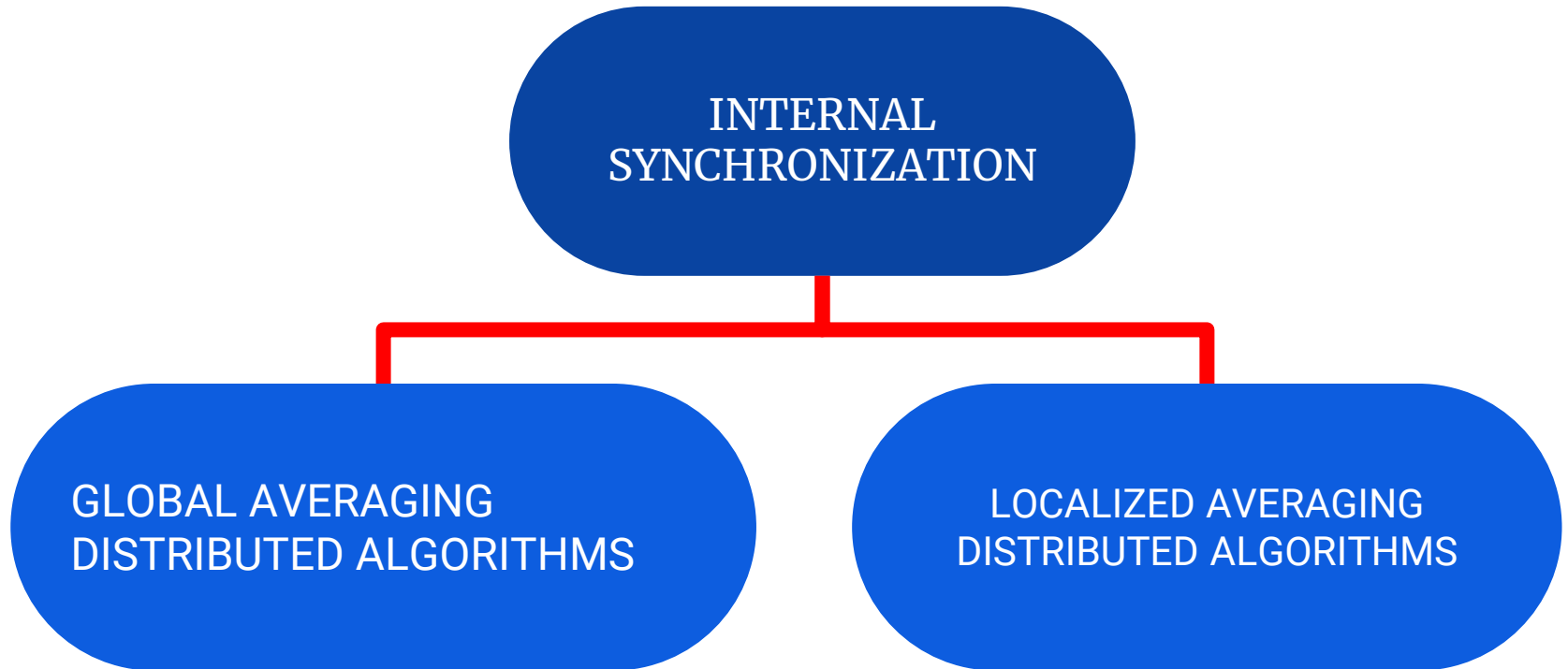
For fault-tolerant average of the clock values of all the computers (including its own) is considered

# CENTRALIZED ALGORITHMS

Centralized clock synchronization algorithms suffer from two major drawbacks:

1. Subject to single-point failure. This makes the system unreliable.
2. From a scalability point of view it is generally not acceptable to get all the time requests serviced by a single time server. In a large system, such a solution puts a heavy burden on that one process.

# DISTRIBUTED ALGORITHMS



# GLOBAL AVERAGING DISTRIBUTED ALGORITHMS

- ❖ The clock process at each node broadcasts its local clock time in the form of a special "resync" message
- ❖ Parameter depends on total number of nodes in the system, the maximum allowable drift rate, and so on
- ❖ These broadcasts will not happen simultaneously from all nodes

# GLOBAL AVERAGING DISTRIBUTED ALGORITHMS

- ❖ The clock process estimates the skew of its clock with respect to each of the other nodes on the basis of the times at which it received resync messages.
- ❖ Fault-tolerant average of the estimated skews is computed



# GLOBAL AVERAGING DISTRIBUTED ALGORITHMS

## Drawbacks

1. Network to support broadcast facility and
2. Large amount of message traffic generated

# LOCALIZED AVERAGING DISTRIBUTED ALGORITHMS.

- ❖ The nodes of a distributed system are logically arranged in some kind of pattern, such as a ring or a grid.
- ❖ Periodically, each node exchanges its clock time with its neighbors in the ring, grid, or other structure
- ❖ Sets its clock time to the average of its own clock time and the clock times of its neighbors

# CASE STUDY

Two popular services for synchronizing clocks

Distributed Time Service (DTS)

Network Time Protocol (NTP).

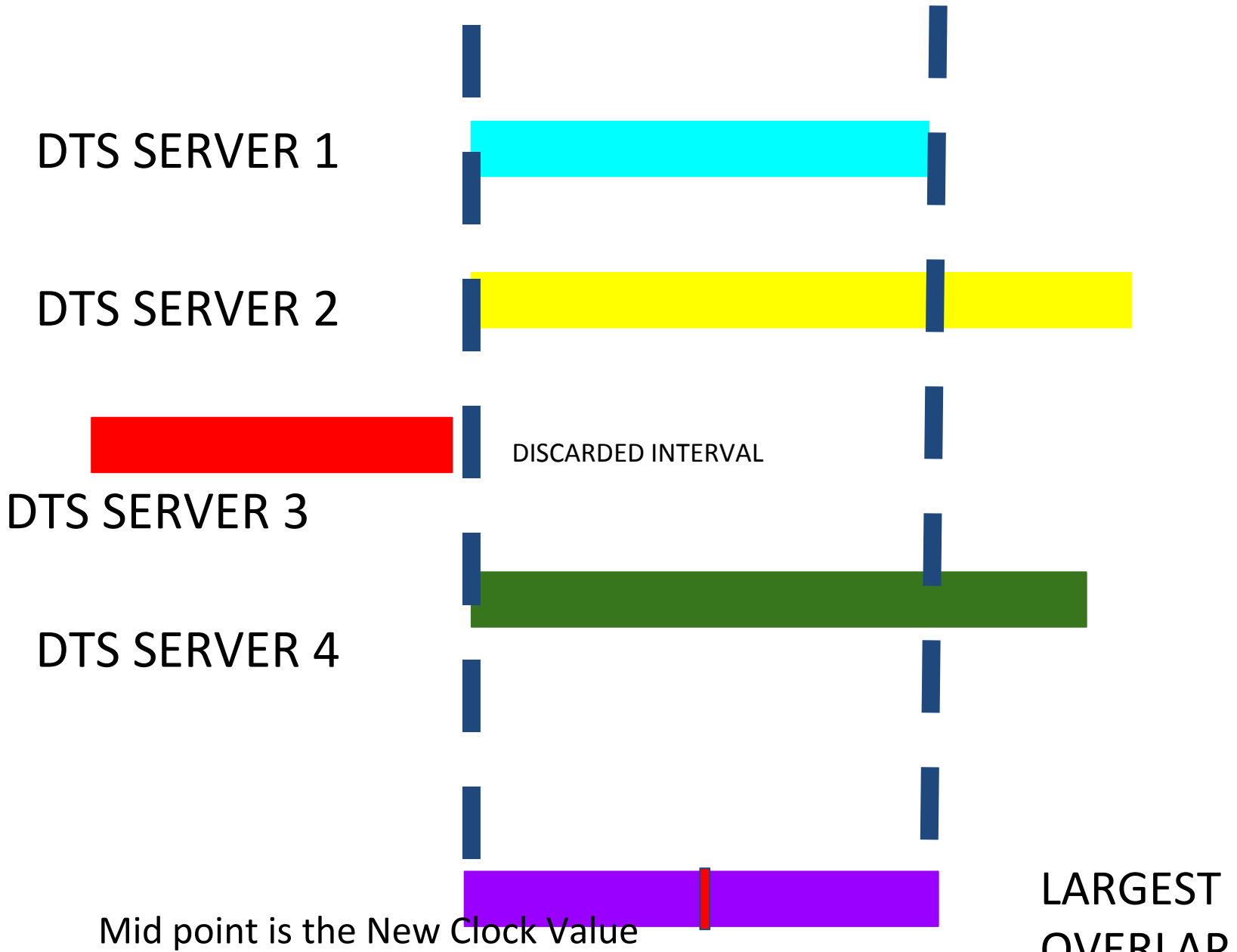
DTS is a component of DCE

DTS does not define time as a single value.

On each DTS client node a process runs called a **DTS clerk** which requests for time information

DTS clerk synchronizes its local clock

It initiates resynchronization by doing an RPC with all the DTS servers on its LAN requesting for the time.



# EVENT ORDERING

it is necessary to ensure that all events that occur in a distributed system be totally ordered in a manner that is consistent with an observed behavior

logical clocks for ordering of events based on the happened-before relation.

# HAPPENED BEFORE RELATION

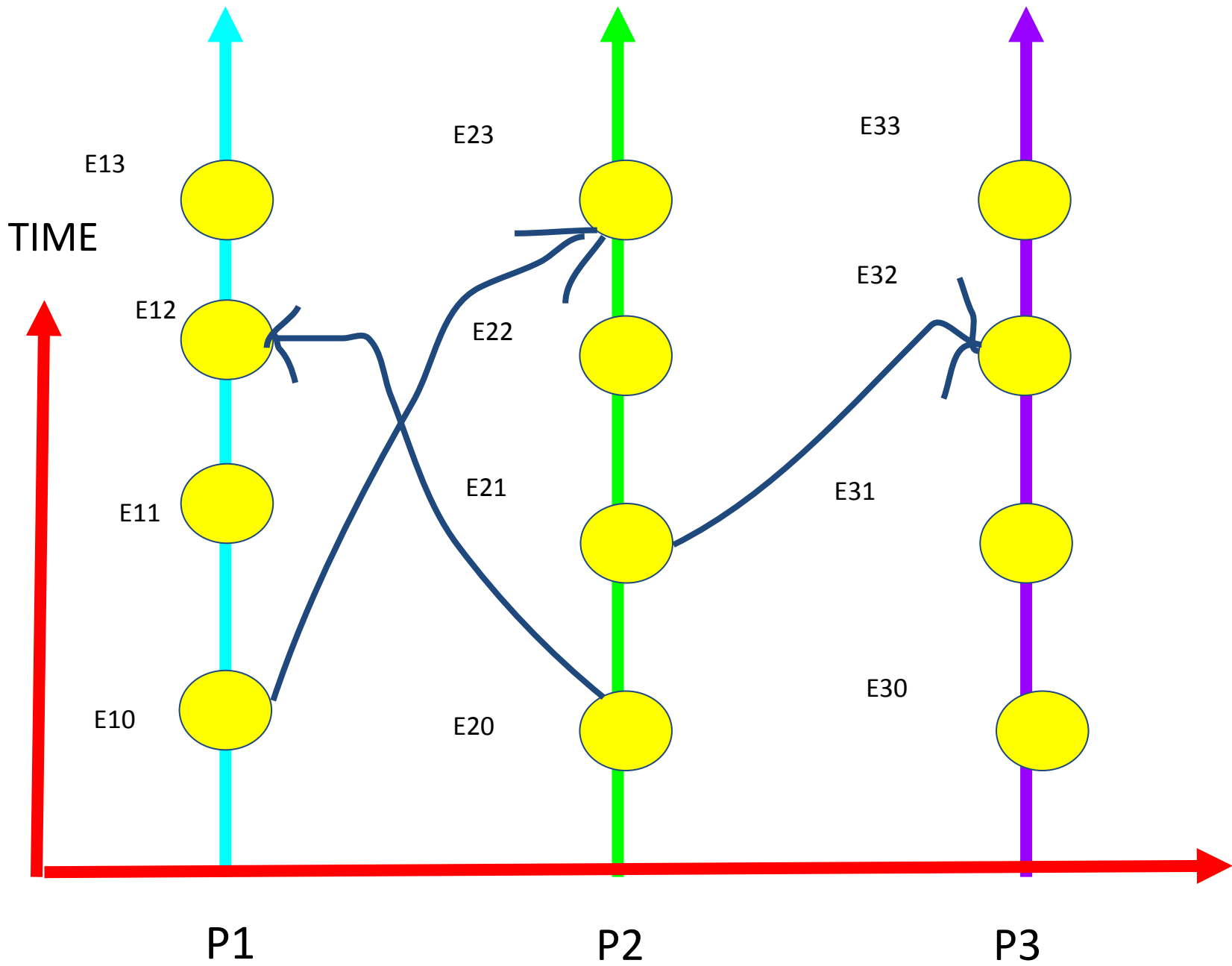
The happened-before relation (denoted by  $\rightarrow$ ) on a set of events satisfies the following conditions:

1. If  $a$  and  $b$  are events in the same process and  $a$  occurs before  $b$ , then  $a \rightarrow b$ .
2. If  $a$  is the event of sending a message by one process and  $b$  is the event of the receipt of the same message by another process, then  $a \rightarrow b$ .
3. If  $a \rightarrow b$  and  $b \rightarrow c$ , then  $a \rightarrow c$ . That is, happened-before is a transitive relation.

# HAPPENED BEFORE RELATION

Two events are concurrent if neither can causally affect the other.

So, the happened-before relation is sometimes also known as the **Relation of causal ordering**.





# LOGICAL CLOCKS CONCEPT

- ❖ in a distributed system
  - the happened-before relation must be defined without the use of globally synchronized physical clocks
  - a common clock or a set of perfectly synchronized clocks are not available.

The logical clocks concept is

is a way **to associate a timestamp**

# LOGICAL CLOCKS CONCEPT

logical clocks must satisfy the following clock condition:

For any two events a and b.

if  $a \rightarrow b$ . then  $C(a) < C(b)$ .

# IMPLEMENTATION OF LOGICAL CLOCKS

For a happened before relation, the following conditions should hold

C1: If  $a$  and  $b$  are two events within the same process  $P_i$  and  $a$  occurs before  $b$ , then

$C_i(a) < C_i(b)$ .

C2: If  $a$  is the sending of a message by process  $P_i$  and  $b$  is the receipt of that message

by process  $P_j$ , then  $C_i(a) < C_j(b)$ .

C3: A clock  $C$ , associated with a process  $P_i$  must always go forward, never backward. That is, corrections to time of a logical clock must always be made by adding a positive value to the clock, never by subtracting value.

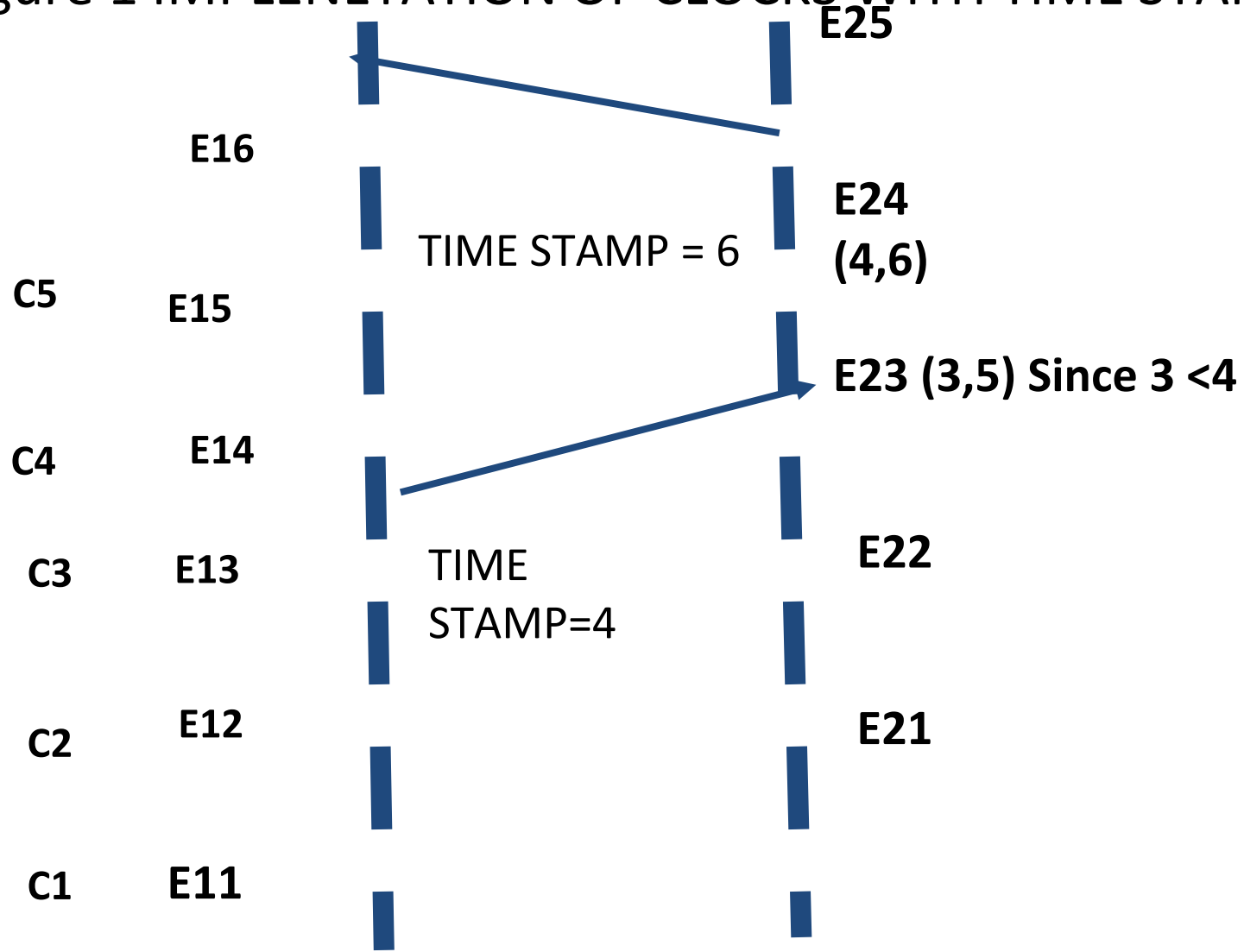
# IMPLEMENTATION OF LOGICAL CLOCKS

To meet conditions C1, C2, and C3, Lamport's algorithm uses the following implementation rules:

**IR1:** Each process  $P_i$  increments  $C$ , between any two successive events.

**IR2:** If event  $a$  is the sending of a message  $m$  by process  $P_i$  the message  $m$  contains a timestamp  $T_m$  and upon receiving the message  $m$  a process  $P_j$  sets  $C$ , greater than or equal to its present value but greater than  $T_m$

Figure 1 IMPLEMENTATION OF CLOCKS WITH TIME STAMP



# TOTAL ORDERING OF EVENTS

the events are ordered by the times at which they occur

when the clocks of both

processes show exactly the same time (say 100), both events will have a timestamp of 100

the time stamps associated with events a and b will be 100.001 and 100.002, respectively, where the process identity numbers of processes P1 and P2 are 001 and 002, respectively.

# MUTUAL EXCLUSION

- A file must not be simultaneously updated by multiple processes.

- use of unit record peripherals such as tape drives or printers must be restricted to a single process at a time.

- Exclusive access to such a shared resource by a process must be ensured.

- Critical sections

(the sections of a program that need exclusive access to shared resources are critical sections)

# MUTUAL EXCLUSION

Requirements:

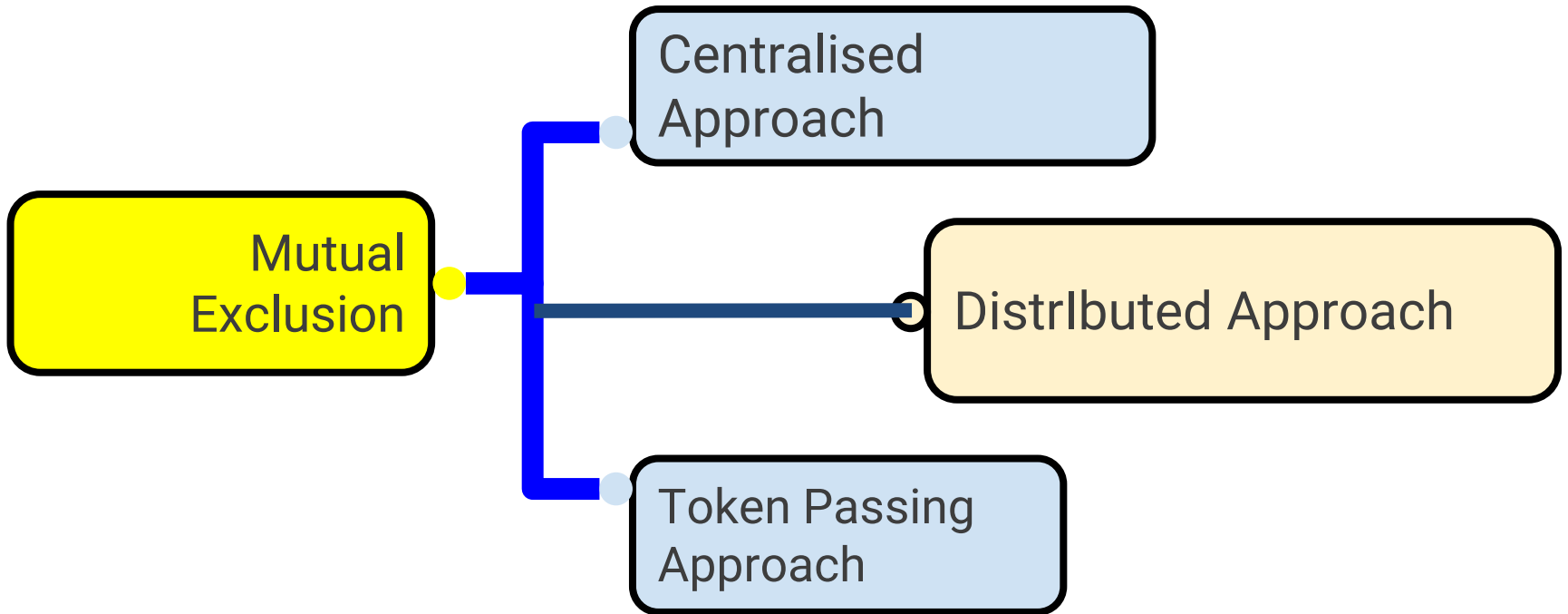
1. Mutual exclusion. At any time only one process should access the resource.

process that has been granted the resource must release it before it can be granted to another process.

2. No starvation. If every process that is granted the resource eventually releases it, every request must be eventually granted.



# MUTUAL EXCLUSION



# CENTRALISED APPROACH

- one of the processes in the system is elected as the coordinator
- Each process that wants to enter a critical section must first seek permission from the coordinator
- If no other process is currently in that critical section, the coordinator can immediately grant permission to the requesting process.
- Grants permission to only one process at a time
- when a process exits the critical section, it must notify the coordinator

# DISTRIBUTED APPROACH

All processes that want to enter the same critical section cooperate with each other before reaching a decision on which process will enter the critical section next.

The first algorithm was presented by Lamport

# LAMPORTS ALGORITHM

When a process wants to enter a critical section, it sends a request message to all other processes. The message contains the following information:

1. The process identifier of the process
2. The name of the critical section that the process wants to enter
3. A unique timestamp generated by the process for the request message

# LAMPORTS ALGORITHM

On receiving a request message, a process either immediately sends back a reply to the sender or defers sending a reply based on the following rules:

1. If the receiver processes currently, it simply queues the request message and defers sending a reply

# LAMPORTS ALGORITHM

2.If the receiver process is currently not executing in the critical section but is waiting for its turn to enter the critical section,

it compares the timestamp in the

If the timestamp of the received request

3. If the receiver process neither is in the critical section nor is waiting for its turn

to enter the critical section, it immediately sends back a reply message.

# LAMPORTS ALGORITHM

Advantages: The algorithm guarantees mutual exclusion because a process can enter its critical section only after getting permission from all other processes.

The algorithm ensures freedom from starvation

Drawbacks

n points of failure

each process should know the identity of all the processes

A process can enter critical section only after communicating with all other processes and getting permission from them.

# LAMPORTS ALGORITHM

Tannenbaum

Modification

sending “Permission denied” rather than  
keeping silent

Improvement

Majority consensus rather than the consensus  
of all other processes for critical section entry



# TOKEN-PASSING APPROACH

Mutual exclusion is achieved by using a single token- a special type of message that entitles its holder to enter a critical section.

The processes in the system are logically organized in a ring structure

The token is circulated from one process to another

# TOKEN-PASSING APPROACH

When a process receives the token, it checks if it wants to enter a critical section and acts as follows:

If it wants to enter a critical section, it keeps the token, enters the critical section, and exits from the critical section

Then passes the token along the ring to its neighbor process.

# TOKEN-PASSING APPROACH

- The process can enter only one critical section when it receives the token.
- If it wants to enter another critical section, it must wait until it gets the token again.
- If it does not want to enter a critical section, it just passes the token
- If none of the processes is interested in entering a critical section, the token simply keeps circulating around the ring

# TOKEN-PASSING APPROACH

## Advantages

Mutual exclusion is guaranteed by the algorithm  
starvation cannot occur

## Drawbacks

1. Process Failure

solution: Detection of a failed process can be easily done by making it a rule that a process receiving the token from its neighbor always sends an acknowledgment message to its neighbor

# TOKEN-PASSING APPROACH

## 2. Loss of Token

Solution: The monitor process periodically circulates a "who has the token?" message on the ring.

This message rotates around the ring from one process to another.

Every process writes its id

After one cycle no entry means token lost, new token generated.

# DEAD LOCKS

Sequence of Events

Request

process first makes a request for the resource

If the requested resource is not available, the requesting process must wait until the it is allocated  
if the system has multiple units the allocation of any unit of the type will satisfy the request

The process can request any no up to maximum

# DEAD LOCKS

Allocate

under system control

Release.

Eg:

two tape drives T1 and T2; two concurrent processes P1 and P2 make requests

1. P1 requests for one tape drive and the system allocates T1 to it.
2. P2 requests for one tape drive and the system allocates T2 to it.
3. P1 requests for one more tape drive and enters a waiting state because no tape drive is presently available.
4. P2 requests for one more tape drive and it also enters a waiting state because no tape drive is presently available.

# DEAD LOCKS

Necessary Conditions for Deadlock

Mutual-exclusion condition

Hold/wait

No Preemption

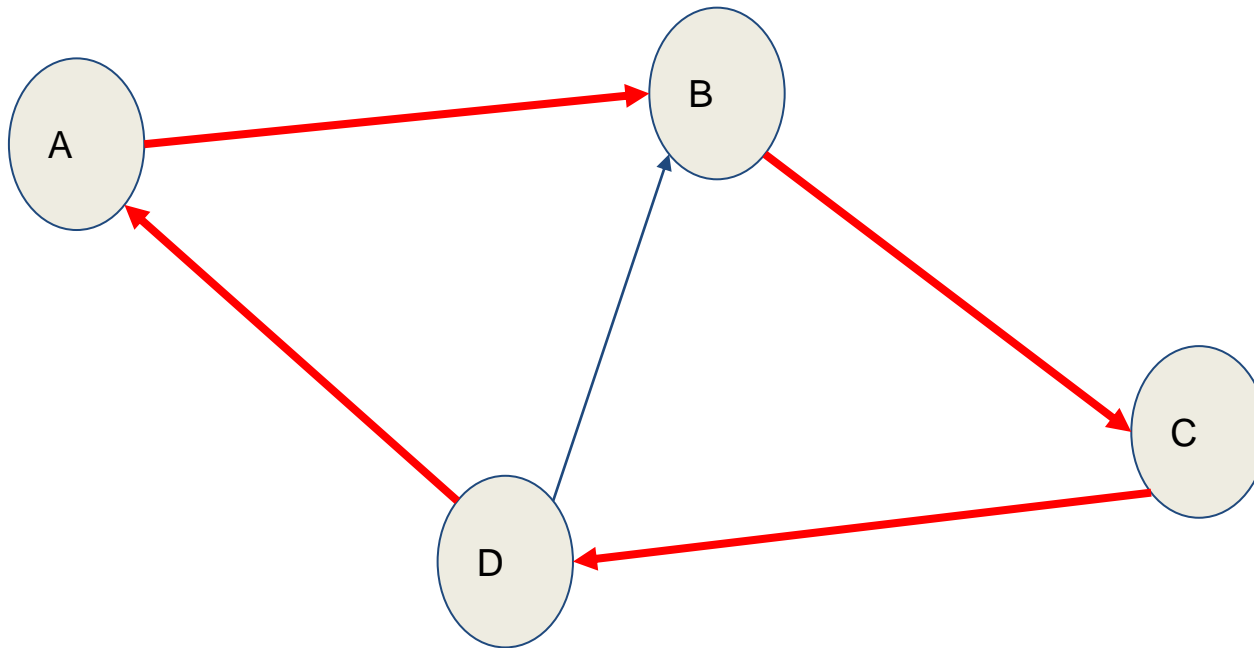
Circular wait

All four conditions must hold simultaneously in a system for a deadlock to occur.



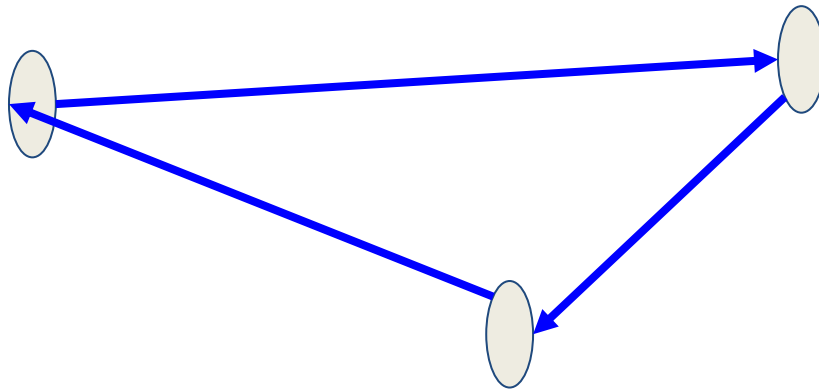
# DEADLOCK MODELING

1. Directed graph. A directed graph is a pair  $(N, E)$ , where  $N$  is a nonempty set of nodes and  $E$  is a set of directed edges. A directed edge is an ordered pair  $(a, b)$ , where  $a$  and  $b$  are nodes in  $N$ .



# DEADLOCK MODELING

2. Path. A path is a sequence of nodes  $(a, b, c, \dots, i, j)$  of a directed graph such that  $(a, b), (b, c), \dots, (i, j)$  are directed edges. Obviously, a path contains at least two nodes.



# DEADLOCK MODELING

3. Cycle. A cycle is a path whose first and last nodes are the same.

4. Reachable set. The reachable set of a node  $a$  is the set of all nodes  $b$  such that a path exists from  $a$  to  $b$ .

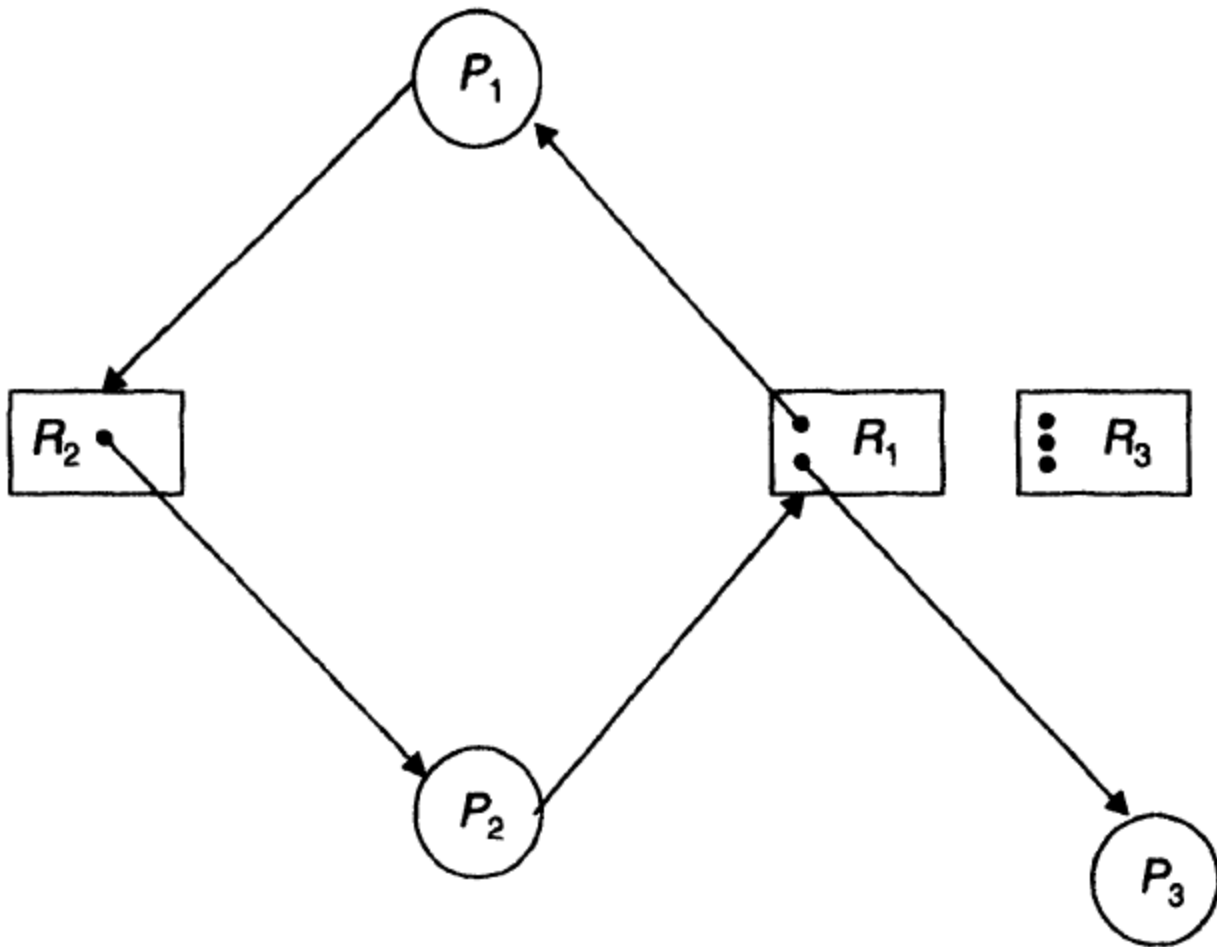
5. Knot. A knot is a nonempty set  $K$  of nodes such that the reachable set of each node in  $K$  is exactly the set  $K$ . A knot always contains one or more cycles.


An example of a directed graph is shown in Figure

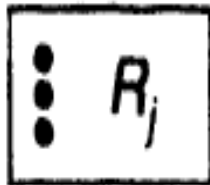
# DEADLOCK MODELING

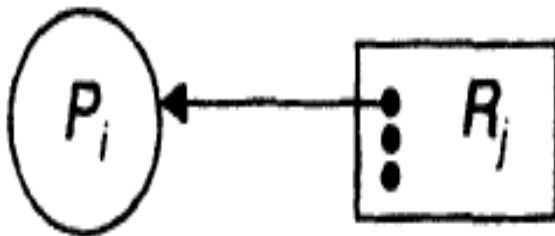
A directed graph, called a resource allocation graph, is used in which both the set of nodes and the set of edges are partitioned into two types.

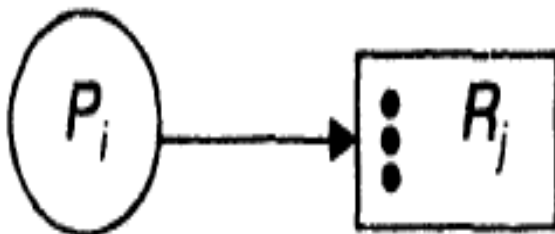
PROCESS NODES, RESOURCE NODES,  
ASSIGNMENT EDGES, REQUEST EDGES



 A process named  $P_i$ .

 A resource  $R_j$  having 3 units in the system.

 Process  $P_i$  holding a unit of resource  $R_j$ .

 Process  $P_i$  requesting for a unit of resource  $R_j$ .

# CYCLE

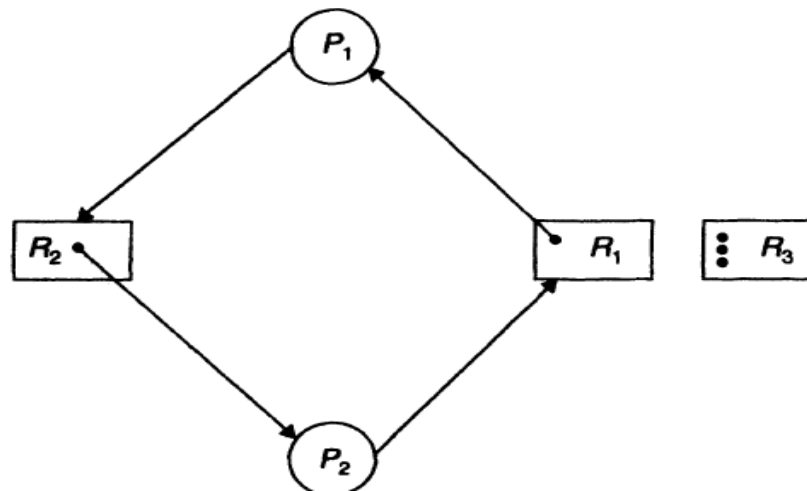
A cycle is a necessary condition for deadlock.

- If there is only a single unit of each resource type involved in the cycle, a cycle is both a necessary and a sufficient condition for a deadlock to exist.
- If one or more of the resource types involved in the cycle have more than one unit, a knot is a sufficient condition for a deadlock to exist.

# NECESSARY AND SUFFICIENT CONDITIONS FOR DEADLOCK

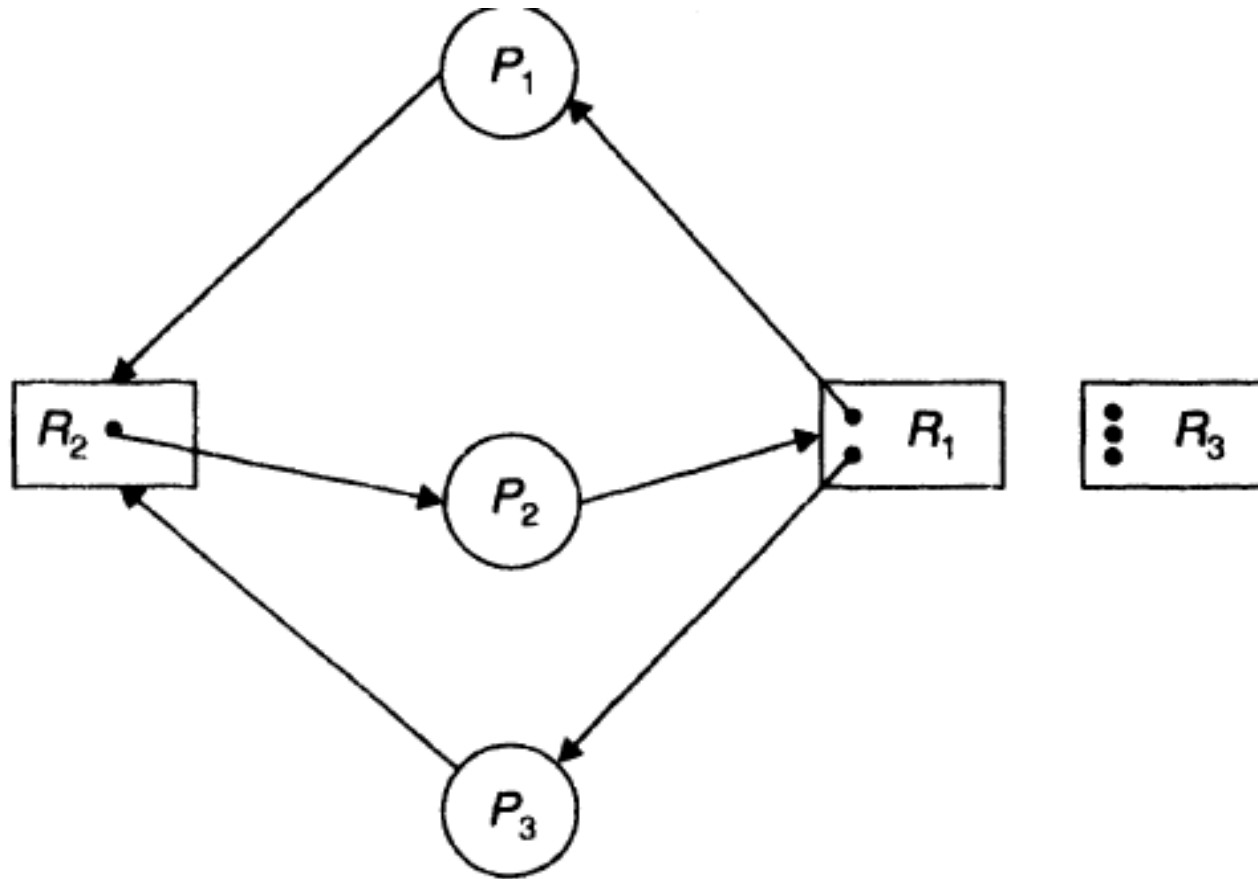
a cycle is a necessary but not sufficient  
condition

Cycle Deadlock





# KNOT DEADLOCK



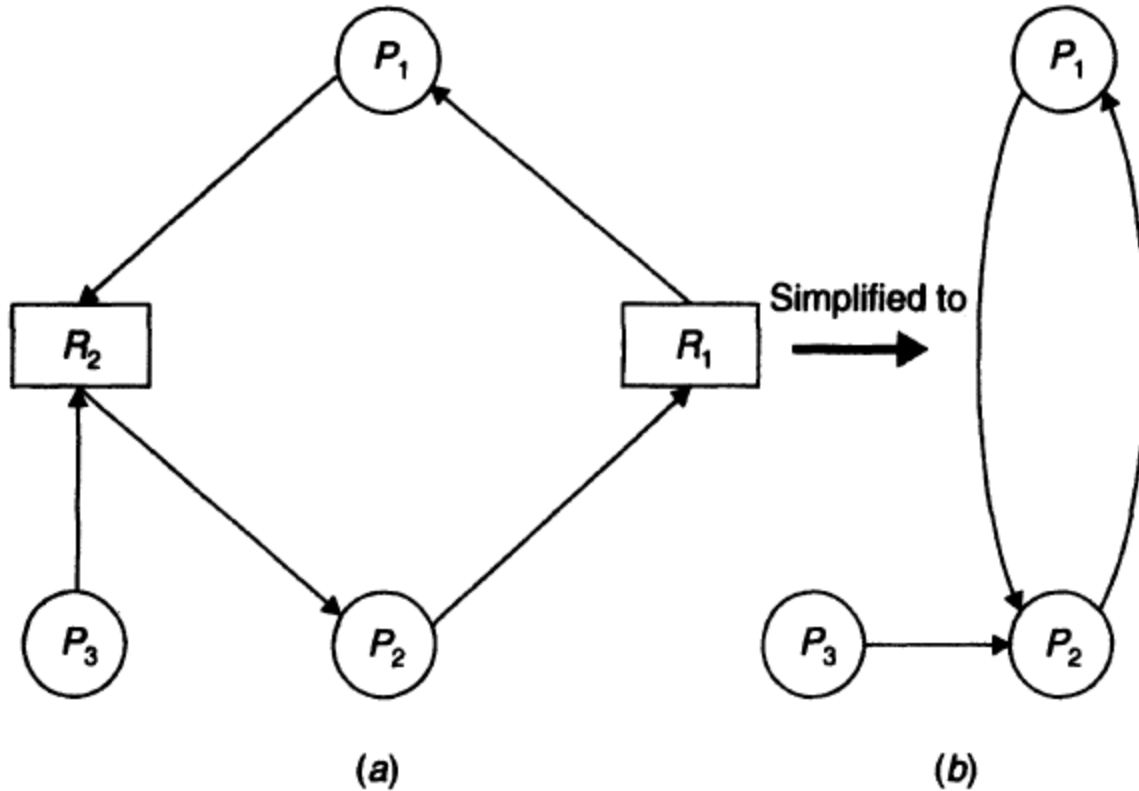
# KNOT

A knot is a set of vertices (processes and resources) such that when starting at any vertex in the knot, paths lead to all vertices in the knot, but to no vertices outside the knot.

Detecting knots:

- start with the generalized resource graph
- remove processes that are not waiting on anything
- remove processes that are waiting on resources that are not fully allocated
- repeat process until no more processes can be removed
- if algorithm is able to remove all processes, no knot is present

# WAIT FOR GRAPH(WFG)



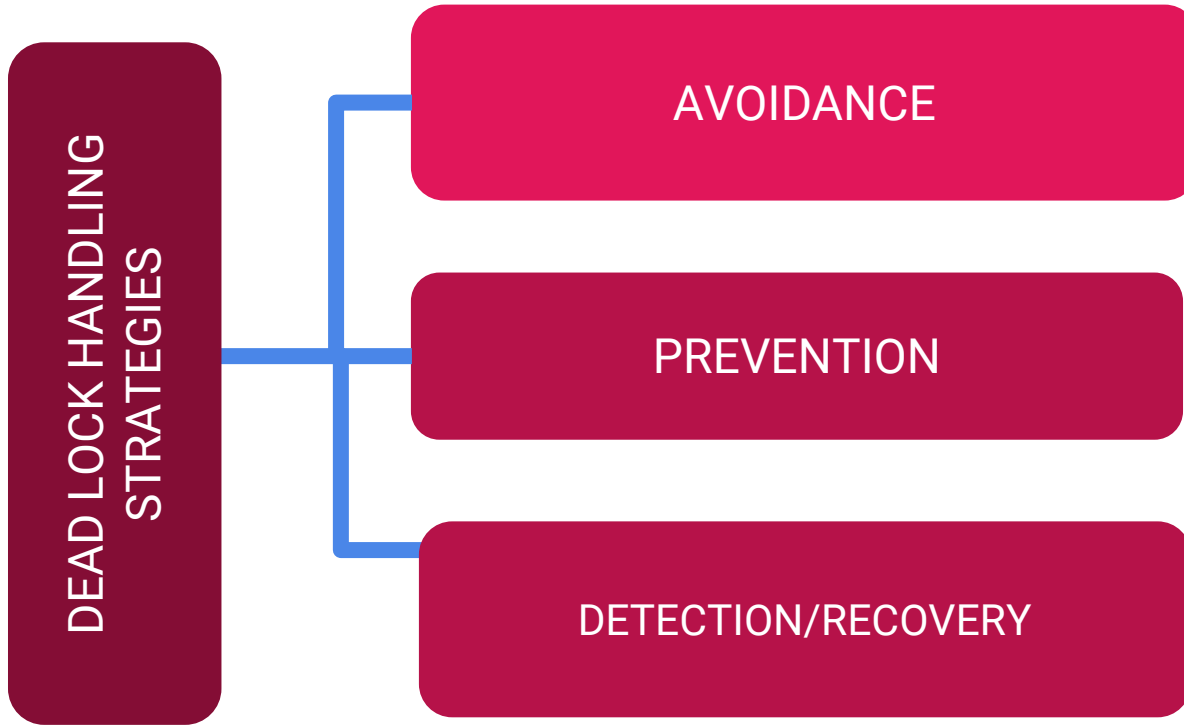
# WFG

Wait For Graph is constructed only when there is only one unit

A cycle in wait for graph(WFG) is a necessary and sufficient condition for DLs

# HANDLING DEADLOCKS IN DISTRIBUTED SYSTEMS

Handling of deadlocks in distributed systems is more complex than in centralized systems because the resources, the processes, and other relevant information are scattered on different nodes of the system



# HANDLING DEADLOCKS

1. Avoidance. Resources are carefully allocated to avoid deadlocks.
2. Prevention. Constraints are imposed on the ways in which processes request resources in order to prevent deadlocks.
3. Detection and recovery. Deadlocks are allowed to occur and a detection algorithm is used to detect them. After a deadlock is detected, it is resolved by certain means.

# HANDLING DEADLOCKS

Two kinds of distributed deadlocks-resource deadlocks and communication deadlocks.

a resource deadlock occurs when two or more processes wait permanently for resources held by each other.

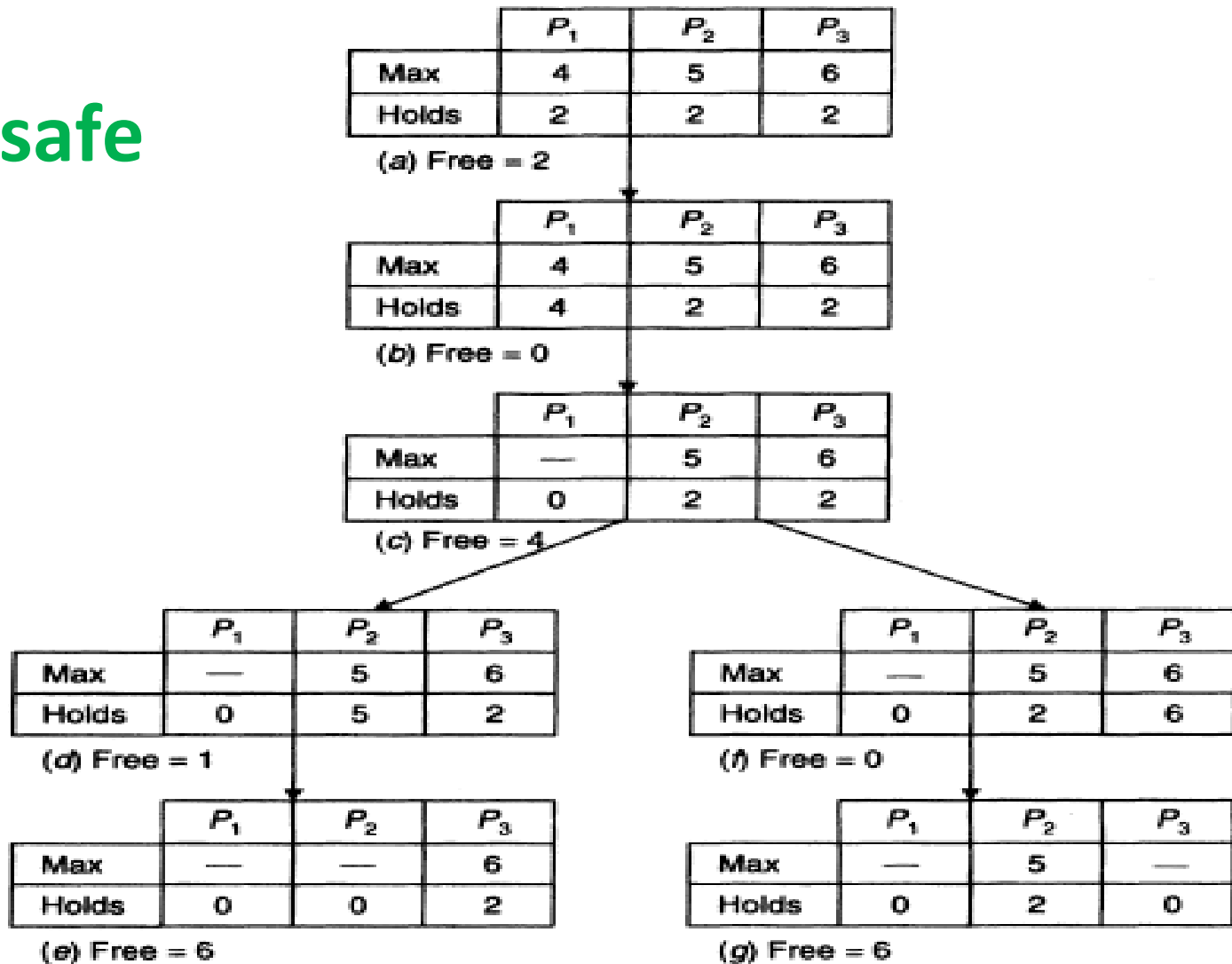
communication deadlock occurs when Processes are blocked waiting for messages from other processes in the set in order to start execution but there are no messages in transit between them.



# DEADLOCK AVOIDANCE

1. When a process requests for a resource, the system simply assumes that the request is granted.
2. The system performs some analysis to decide whether granting the process's request is safe or unsafe.
3. The resource is allocated that it is safe to do so; otherwise the request is deferred.

safe

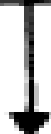


Reference: Pradeep, K. Sinha "Distributed Operating System Concepts and Design"

# unsafe

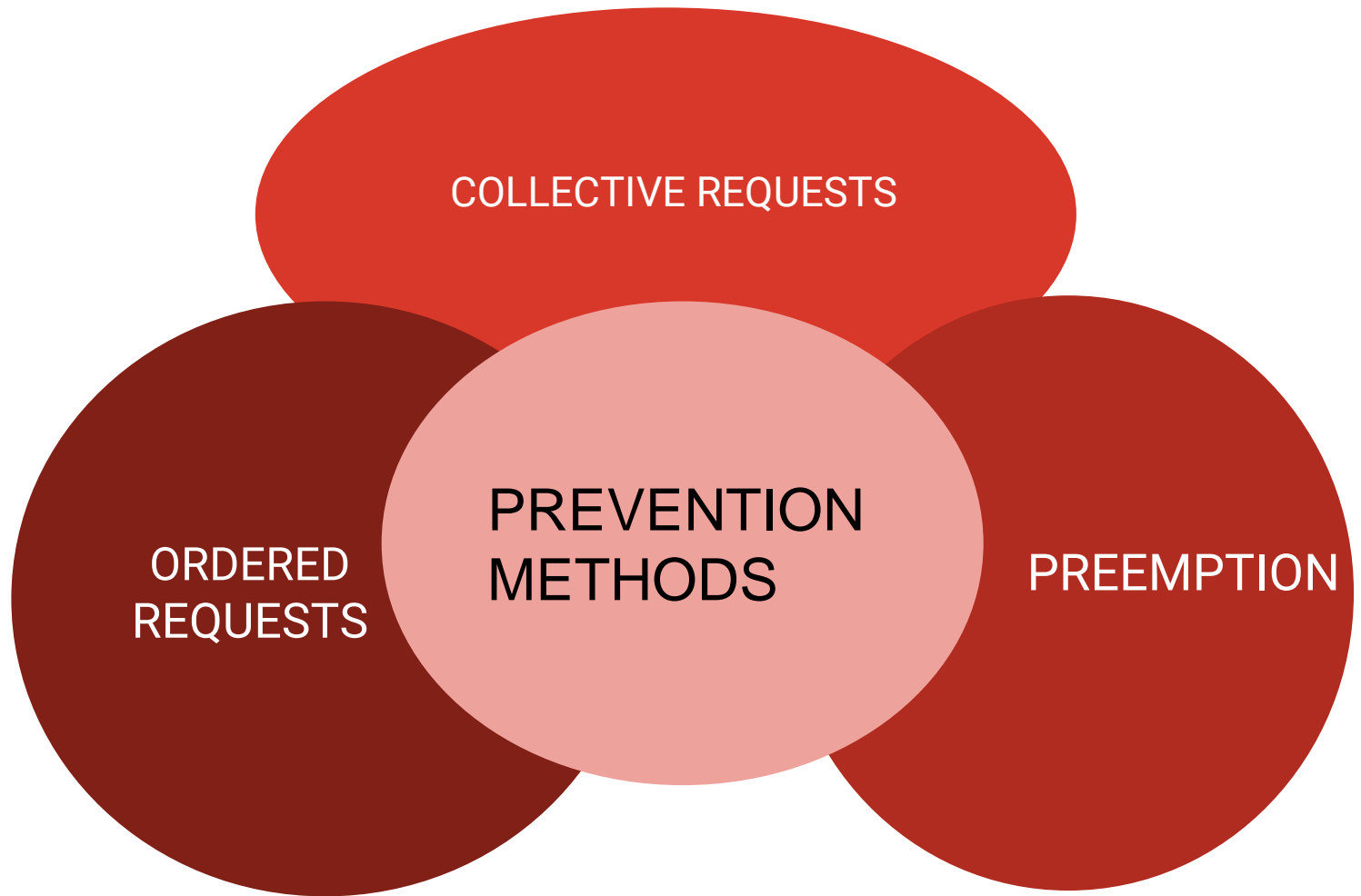
	$P_1$	$P_2$	$P_3$
Max	4	5	6
Holds	2	2	2

(a) Free = 2



	$P_1$	$P_2$	$P_3$
Max	4	5	6
Holds	2	3	2

(b) Free = 1



# COLLECTIVE REQUESTS

1. A process must request all of its resources before it begins execution
2. It requests resources only when it holds no other resources

## Disadvantages

starvation

poor utilization

accounting

# ORDERED REQUESTS

Efficient method  $r1=0$   $r2=1$   $r3=3$

resources are assigned with numbers  $p1=r2$   $p1r3$

process can request only with high number

Problems

wastage of resources

Reordering will require reprogramming of jobs.

# PREEMPTION

if resource un available

1. All the resources held by the process are taken away (preempted) from it and the process is blocked.
2. The process is unblocked when the resource requested by it and the resources preempted from it become available and can be allocated to it.

# PREEMPTION

if resource unavailable

3. the system checks if the requested resource is currently held by a process that is blocked, waiting for some other resource. If so, the requested resource is taken away (preempted) from the waiting process and given to the requesting process.



# PREEMPTION

Rosenkrantz et al proposed the following deadlock prevention schemes

1. Wait-die scheme.
2. Wait-wound scheme

# DETECTION/RECOVERY

Maintaining WFG and searching for cycles in the WFG.

Construct a separate WFG for each site of the system

Convert the resource allocation graph constructed in step 1 to a corresponding WFG by removing the resource nodes and collapsing the appropriate edges.

Take the union of the WFGs of all sites and construct a single global WFG.

# DETECTION/RECOVERY

The main problem is

how to maintain WFG

Three commonly used techniques for organizing the WFG in

a distributed system are centralized, hierarchical, and distributed.

# DETECTION/RECOVERY

The detection depends on the following properties

1. Progress property- all deadlocks must be detected in a finite amount of time.
2. Safety property. If a deadlock is detected, it must indeed exist.

# DETECTION/RECOVERY

Message delays and out-of-date WFGs sometimes cause false cycles to be detected, resulting in the detection of deadlocks that do not actually exist. Such deadlocks are called phantom deadlocks

# DETECTION/RECOVERY

Centralized Approach for Deadlock Detection

INTIMATION OF EDGE ADDED to detect  
deadlocks

Continuous transfer.

Periodic Transfer

Transfer on request

# DETECTION/RECOVERY

Fully Distributed Approaches for Deadlock Detection.  
each site of the system shares equal responsibility for  
deadlock detection.

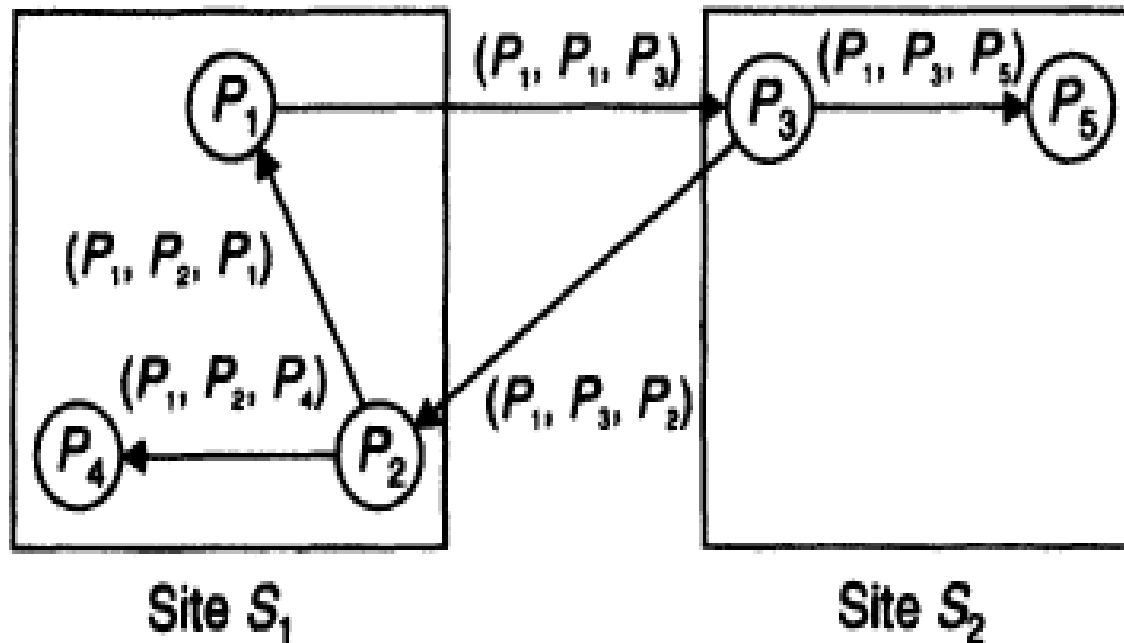
1. WFG Algorithm
2. Probe based Algorithm

WFG construction of wait for Graphs

Probe base means

sending probes from one node to another if resource not  
available

# PROBE BASED-CHANDY MISRA HASS ALGORITHM



Reference: Pradeep,K.Sinha "Distributed Operating System Concepts and Design"



# WAYS FOR RECOVERY FROM DEADLOCK

## Ways for Recovery from Deadlock

- ❑ Asking for operator intervention
- ❑ Termination of process
- ❑ Rollback of process

## Issues:

- ❖ Selection of a victim
- ❖ cost Minimization
- ❖ Transaction mechanism

# ELECTION ALGORITHMS

## Co-ordinator Election

Election algorithms are based on

1. Each process in the system has a unique priority number.
2. the process having the highest priority number among the currently active processes is elected as the coordinator.
3. On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

# THE BULLY ALGORITHM

Proposed by Garcia-Molina

A process sends a request message to the coordinator

Does not receive a reply within a fixed timeout period

it is assumed that the coordinator has failed.

Election Message

Alive Message

P1, P2, P3, P4, and P5 and their priority numbers are

1, 2, 3, 4, and 5 respectively

# THE BULLY ALGORITHM

1. Obviously, P5 is the coordinator in the starting state.
2. Suppose P5 crashes.
3. Process P3 sends a request message to Ps and does not receive a reply within the fixed timeout period.
4. Process P3 assumes that P5 has crashed and initiates an election by sending an election message to P4 and P5

# THE BULLY ALGORITHM

5. When P4 receives P3's election message, it sends an alive message to P3'

informing that it is alive and will take over the election activity. Process Ps cannot respond to P3's election message because it is down.

6. Now P4 holds an election by sending an election message to Ps-

# THE BULLY ALGORITHM

Process P5 does not respond to P4's election message because it is down, and therefore, P4 wins the election and sends a coordinator message to P1, P2, and P3, informing them that from now on it is the new coordinator.

Now suppose P2 recovers from failure and initiates an election by sending an election message to P3, P4, and P5.

# A RING ALGORITHM

All the processes in the system are organized in a logical ring.

The ring is unidirectional in the

All messages related to the election algorithm are always passed only in one direction

If the successor of the sender process is down, the sender can skip over the successor, or the one after that, until an active member is located. The algorithm works as follows.

# A RING ALGORITHM

When a process (say  $P_i$ ) sends a request message to the current coordinator and does not receive a reply within a fixed timeout period, it assumes that the coordinator has crashed. Therefore it initiates an election. On receiving the election message, the successor appends its own priority number to the message and passes it on to the next active member in the ring.



# A RING ALGORITHM

This member appends its own priority number to the message and forwards it to its own successor.

The election message circulates over the ring from one active process to another and returns back to process  $P_i$ .

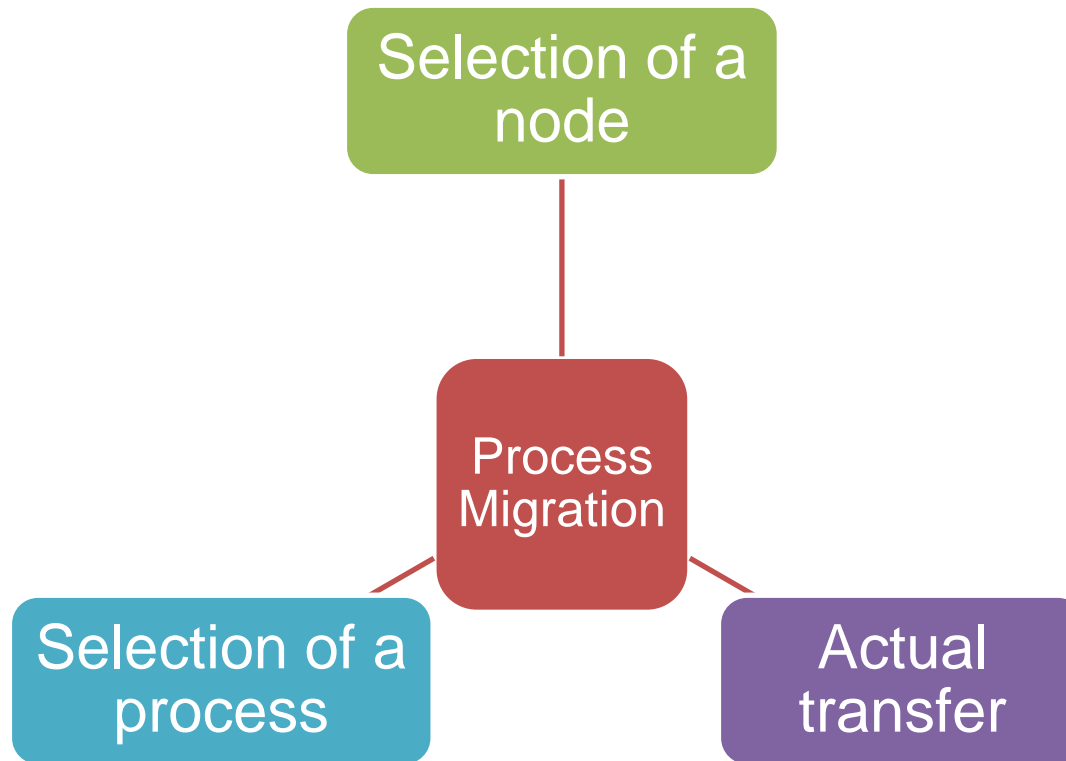
Process  $P_i$  recognizes the message as its own election message.

it elects the process having the highest priority number as the new coordinator.

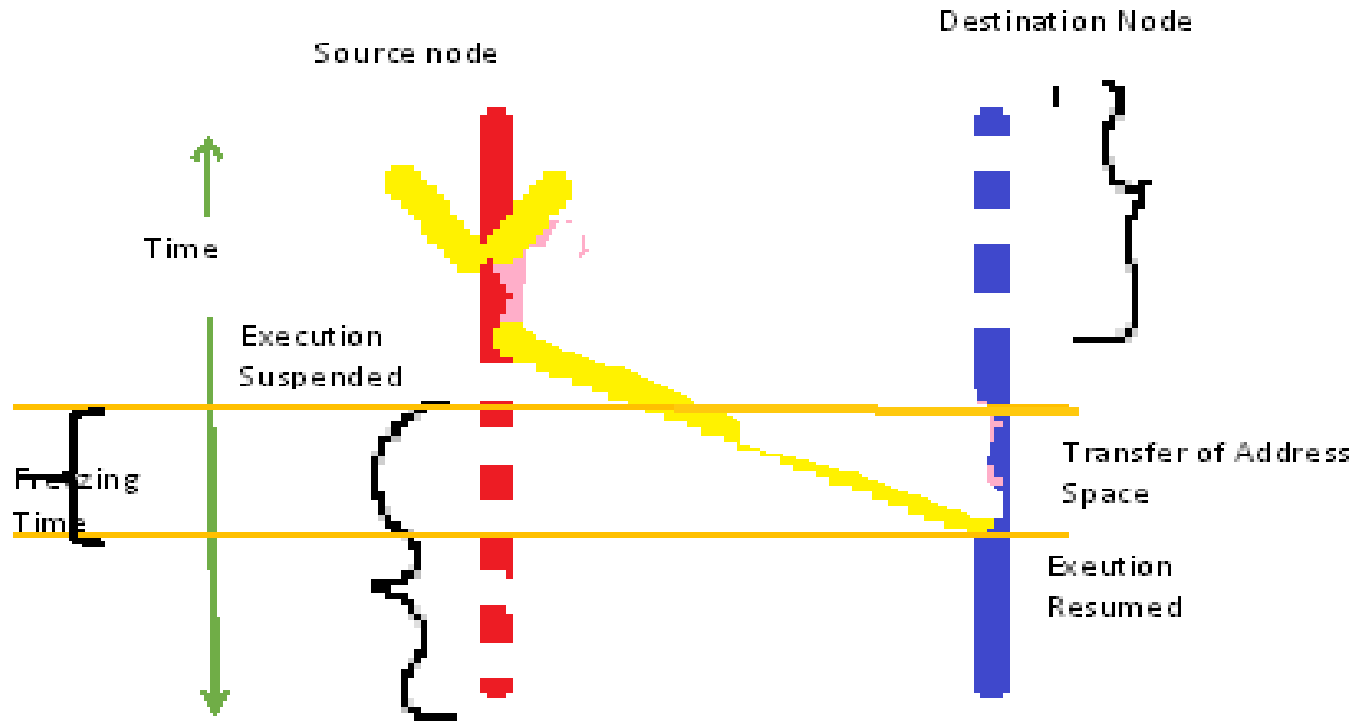
# PROCESS MIGRATION

Movement of a process from its current location to the processor to which it has been assigned.

1. Selection of a process that should be migrated
2. Selection of the destination node to which the selected process should be migrated
3. Actual transfer of the selected process to the destination node



# PROCESS MIGRATION



# Desirable Features of a Good Process Migration Mechanism

## Non-Preemptive Process Migration:

A process may be migrated either before it starts executing on its source node

## Preemptive process migration:

The transfer takes place during the course of its execution.

# Desirable Features of a Good Process Migration Mechanism

## Transparency

1. Object access level.
2. System call and interprocess communication level.

## Minimal Interference

## Minimal Residual Dependencies

## Efficiency

1. The time required for migrating a process
2. The cost of locating an object
3. The cost of supporting remote execution once the process is migrated

# Desirable Features of a Good Process Migration Mechanism

## Robustness

Failure of a node other than the one on which a process is currently running should not in any way affect the accessibility or execution of that process.

## Communication between Co-Processes of a Job

# PROCESS MIGRATING MECHANISMS

Major sub activities involved in process migration

1. Freezing the process on its source node and restarting it on its destination node
2. Transferring the process's address space from its source node to its destination node
3. Forwarding messages meant for the migrant process
4. Handling communication between cooperating processes



# MECHANISMS FOR FREEZING AND RESTARTING A PROCESS

Immediate and Delayed Blocking of the Process.

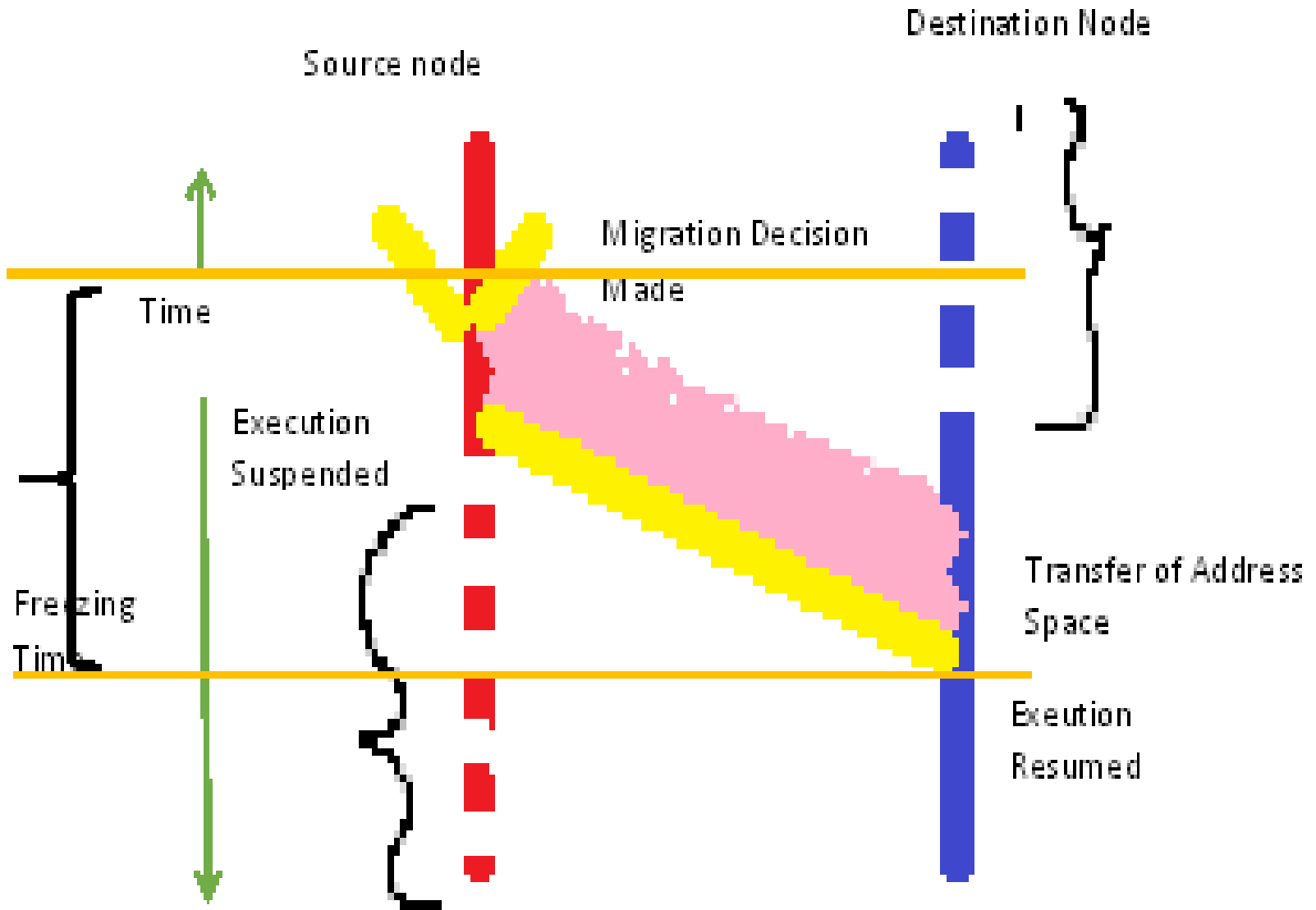
Fast and Slow I/O Operations

Information about Open Files

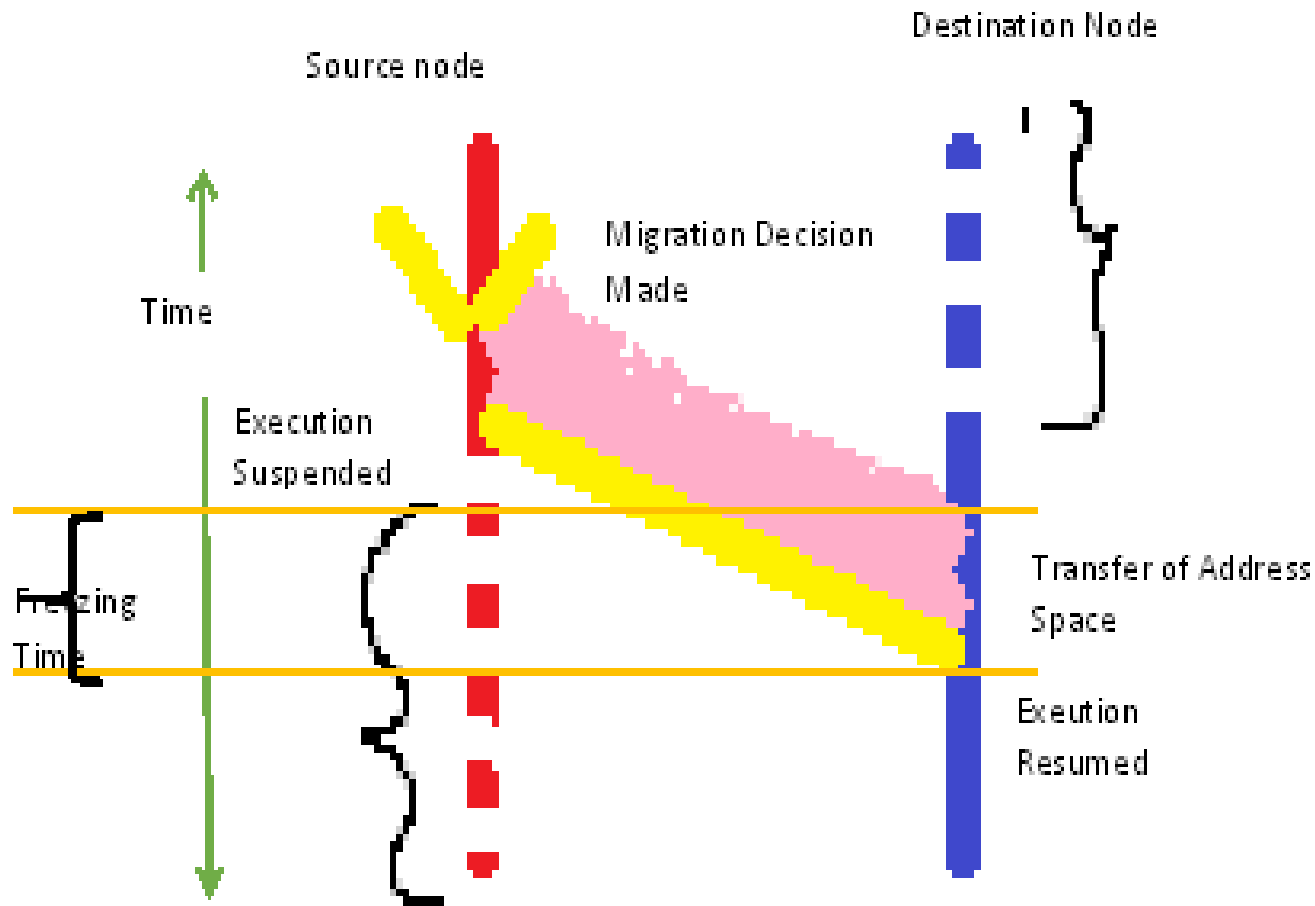
Reinstating the Process on its Destination Node

Address Space Transfer Mechanisms

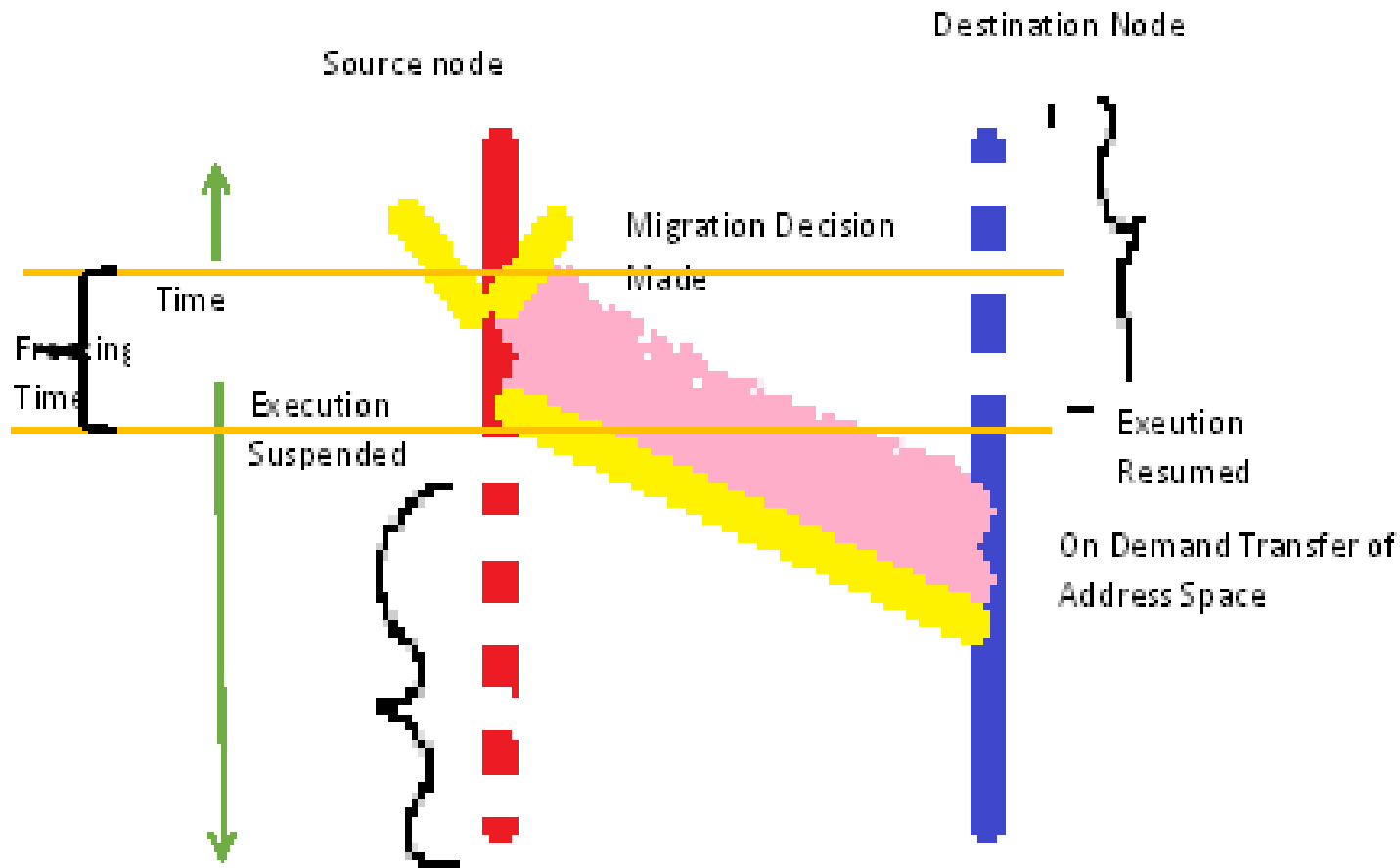
# TOTAL FREEZING



# PRE TRANSFER MECHANISM



# TRANSFER ON REFERENCE



# MESSAGE-FORWARDING MECHANISMS

Type 1: Messages received at the source node after the process's execution has been stopped on its source node and the process's execution has not yet been started on its destination node

Type 2: Messages received at the source node after the process's execution has started on its destination node

Type 3: Messages that are to be sent to the migrant process from any other node after it has started executing on the destination node

# MECHANISM OF RESENDING THE MESSAGE

In this method, messages of types 1 and 2 are returned to the sender as not deliverable or are simply dropped, with the assurance that the sender of the message is storing a copy of the data and is prepared to retransmit it.

# ORIGIN SITE MECHANISM.

The process identifier of these systems has the process's origin site embedded in it

Each site is responsible for keeping information about the current locations of all the processes created on it.

A process's current location can be simply obtained by consulting its origin site.

The origin site then forwards the message to the process's current location.

# LINK TRAVERSAL MECHANISM.

To redirect the messages of types 2 and 3, a forwarding address known as link is left at the source node pointing to the destination node of the migrant process using two components

1. Process id
2. Last known location



# LINK UPDATE MECHANISM.

The source node sends link-update messages to the kernels controlling all of the migrant process's communication partners.

These link update messages tell the new address of each link held by the migrant process and are acknowledged.

# MECHANISMS FOR HANDLING COPROCESSES

Necessity to provide communication between a process (parent) and its sub processes (children), which might have been migrated and placed on different nodes

**Two different mechanisms :**

Disallowing Separation of Co-processes

Home Node or Origin Site Concept

# PROCESS MIGRATION IN HETEROGENEOUS SYSTEMS

## Data translation

Maguire and Smith [1988] proposed the use of the external data representation

A standard representation is used for the transport of data, and each processor

needs only to be able to convert data to and from the standard form

External data representation format is called serializing, and the reverse process is called deserializing.

# ADVANTAGES OF PROCESS MIGRATION

Reducing average response time of processes.

Speeding up individual jobs

Gaining higher throughput

Utilizing resources effectively

Reducing network traffic

Improving system reliability

Improving system security

# THREADS

Way to improve application performance through parallelism

In traditional operating systems the basic unit of CPU utilization is a process. Each process has its own program counter, its own register states, its own stack, and its own address space.

in operating systems with threads facility, the basic unit of CPU utilization is a thread. In these operating systems, a process consists of an address space and one or more threads of control

# THREADS

Each thread of a process has its own program counter, its own register states, and its own stack.

share the same address space.

share the same global variables.

share the same set of operating system resources, such as openfiles, child processes, semaphores, signals, accounting information, and so on.

# THREADS

Due to the sharing of address space, there is no protection between the threads of a process.

But a process is always owned by a single user.

If protection is required between two threads of a process, it is preferable to put them in different processes, instead of putting them in a single process.

# THREADS

- ❑ Thread can be in anyone of several states: running, blocked, ready, or terminated.
- ❑ Threads are referred to as lightweight processes
- ❑ Traditional processes are referred to as heavyweight processes.
- ❑ A process having a single thread corresponds to a process of a traditional operating system

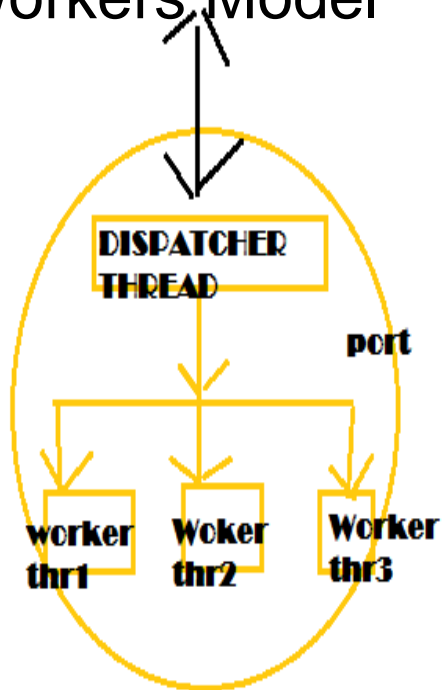


# MOTIVATIONS FOR USING THREADS

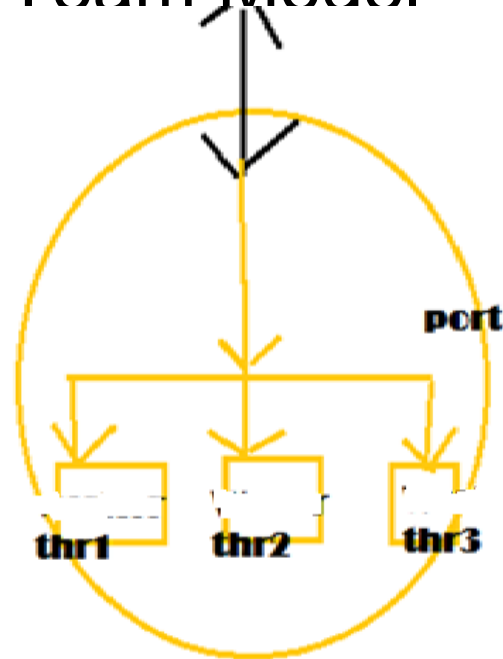
1. The overheads involved in creating a new process are greater than those of creating a new thread within a process.
2. Switching between threads sharing the same address space is cheaper than switching between processes that have their own address spaces.
3. Threads allow parallelism
4. Resource sharing can be performed

# MODELS FOR ORGANIZING THREADS

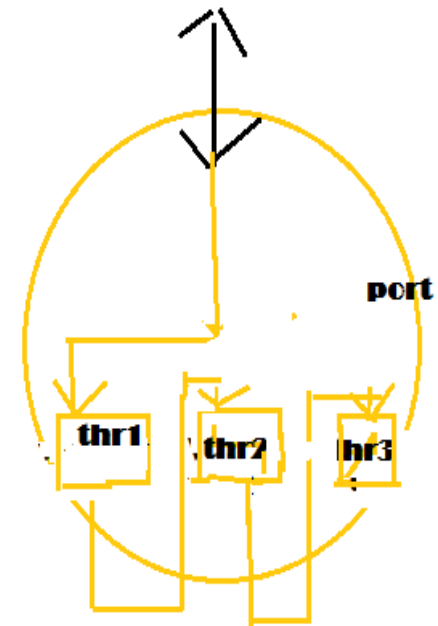
Dispatcher-workers Model



Team Model



Pipeline Model



# ISSUES IN DESIGNING A THREADS PACKAGE

## Threads Creation

Static or dynamic

## Threads Termination

Destroy itself or killed from outside

## Threads Synchronization

Mutex variables

Conditional variables

## Threads Scheduling

Priority assignment facility.

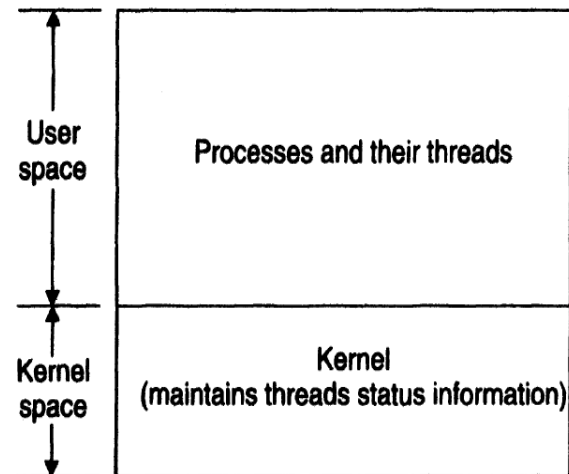
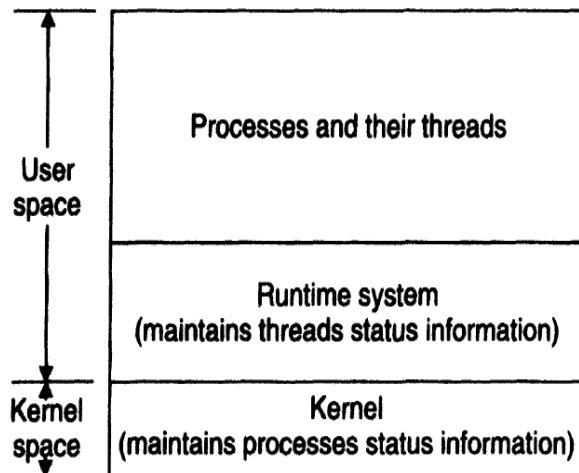
Flexibility to vary quantum size dynamically

Handoff scheduling

Affinity scheduling.

# IMPLEMENTING A THREADS PACKAGE

can be implemented either in user space or in the kernel



Reference: Pradeep, K. Sinha "Distributed Operating System Concepts and Design"